Crypto for PETs – Part 3

Jorge Cuellar

WS 18-19

Processing data securely in the Cloud

Assume that a "client" is working in a cloud ("server")

- and some data fields are numbers (sensitive values)
  - say, amounts of money in Euro
  - they must be maintained encrypted
    - so that the cloud provider (the server) or any other in the cloud
    - is not able to read the cleartext

But the client wants to *process the data in the cloud*

- That means: upload a program to the could,
  - do the arithmetic processing there
  - fetch back the data
    - and — only then — decrypt it

Is it possible to "calculate with encrypted data"?

Homomorphic Encryption

Homomorphic encryption

allows

- addition and/or multiplication
- to be carried out on the encrypted values
  - when the result is decrypted, it yields the same result
  - as the same calculation on the unencrypted inputs:
- $\mathscr{D}(\mathscr{E}(v_1) \circ \mathscr{E}(v_2)) = (v_1 \circ v_2)$

In other words,

- $(\mathscr{E}(v_1) \circ \mathscr{E}(v_2))$ is one encryption of $(v_1 \circ v_2)$

Fully-homomorphic encryption (for both, addition and mult)

- is ongoing research

Recall: RSA with public parameter $n = p \cdot q$

p, q: two random secret primes

- d: the *public key* is a random number: $1 < d < n - 1$
- e: the *private key* is a number with: $d \cdot e \equiv_{(p-1)(q-1)} 1$

Message m is encrypted as

- $\mathcal{E}(m) := m^e \in \mathbb{Z}_n^*$

and $c = \mathcal{E}(m)$ is decrypted via

- $D(c) := c^d \in \mathbb{Z}_n^*$

Homomorphic Encryption

RSA is multiplicatively homomorphic:

If you encrypt two numbers separately,

- using the same secret key,
    - multiply the ciphertexts, then decrypt the result,
- you get the same result that you would get
    - if you multiplied the two original numbers

But RSA is not homomorphic for addition

Homomorphic Encryption

RSA is Multiplicatively Homomorphic

RSA is multiplicative homomorphic:

- $\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) = \mathcal{E}(m_1 \cdot m_2)$

Given

- $c_i = \mathcal{E}(m_i) = m_i^e \bmod N$

$$
\begin{aligned}
c_1 &= m_1^e &\bmod N\\
c_2 &= m_2^e &\bmod N\\
c_1 \cdot c_2 &= m_1^e \cdot m_2^e &\bmod N\\
&= (m_1 \cdot m_2)^e &\bmod N
\end{aligned}
$$

Homomorphic encryption for Aggregation

Homomorphic cryptosystems are used to create aggregated data

- calculate some statistics (averages, sums, etc) on personal data
  - that hide (in some cases) the values of the sensitive personal data

Homomorphic encryption can be used

- for example secure voting systems
- for private information retrieval schemes
- and many more

El Gamal Encryption

Public parameters:

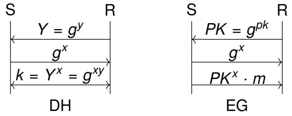
- G, a group,
- $|G|$, the order of the group,
- g, a generator of G

The public key is

- $PK = g^{pk} \in G$ for some secret private key $pk \in \{1, \ldots, |G| - 1\}$

To encrypt a message $m \in G$,

- generate a random $x \in \{1, \ldots, |G|\}$
- $\mathcal{E}(m) := (g^x, PK^x \cdot m)$



El Gamal is a homomorphic encryption

Given encryptions

- $\mathcal{E}(m_1) = (g^{x_1}, PK^{x_1} \cdot m_1)$
- $\mathcal{E}(m_2) = (g^{x_2}, PK^{x_2} \cdot m_2)$

Then the pointwise product of those two encrypted messages

- $\mathcal{E}(m_1) \cdot \mathcal{E}(m_2) =$
  - $(g^{x_1}, PK^{x_1} \cdot m_1) \cdot (g^{x_2}, PK^{x_2} \cdot m_2) :=$
  - $(g^{x_1} \cdot g^{x_2}, PK^{x_1} \cdot m_1 \cdot PK^{x_2} \cdot m_2)$
- is an encryption of the product $(m_1) \cdot (m_2)$

Proof:

- If we multiply two messages componentwise, we get

$$(g^{x_1}, PK^{x_1} \cdot m_1) \cdot (g^{x_2}, PK^{x_2} \cdot m_2) = (g^{x_1+x_2}, PK^{x_1+x_2} \cdot m_1 \cdot m_2) = (g^{x}, PK^{x} \cdot (m_1 \cdot m_2))$$

El Gamal is homomorphic with respect to multiplication

Oblivious Transfer

An oblivious transfer protocol (OT)

- is a type of query-response protocol

The "client" or "receiver" asks for a piece of information

- say: an entry of a DB

The "server" or "sender" responds with the information

- or with nothing
  - BUT: he remains oblivious (= unaware, unconscious) about
    - the content of the query
    - what piece (if any) has been transferred

In some variants of OT

- it is not a query-response protocol
  - simply a "send" protocol
- where a sender transfers one of
  - piece of information to a receiver out of a set
    - without knowing which one

1-2 Oblivious Transfer

Recall first D-H:

- A chooses $a \leftarrow \mathbb{Z}_p$, $A = g^a$
- B chooses $b \leftarrow \mathbb{Z}_p$, $B = g^b$

$$A \xrightarrow{\;A\;} B$$

$$A \xleftarrow{\;B\;} B$$

$$A \xrightarrow{\;e \leftarrow E_k(m)\;} B$$

Where E_k is encryption with the key k known to both: $k = B^a = A^b$

- Observe that Alice can also derive
  - $\widehat{B}^a = (\frac{B}{A})^a$
    - but Bob cannot compute it this group element (assuming CDH)

1-2 Oblivious Transfer

Consider the following D-H variant:

- A chooses $a \leftarrow \mathbb{Z}_p$, $A = g^a$
- B chooses $b \leftarrow \mathbb{Z}_p$, $B = g^b$

$$A \xrightarrow{\quad A \quad} B$$

$$A \xleftarrow{\quad \widetilde{B}=AB \quad} B$$

$$A \xrightarrow{\quad e \leftarrow E_k(m) \quad} B$$

Where E_k is encryption with the key k known to both:

- $k = (\frac{\widetilde{B}}{A})^a = (\frac{AB}{A})^a = A^b$

Oblivious Transfer

- A has two messages m_0, m_1
- B wants message i
- As above
  - A chooses $a \leftarrow \mathbb{Z}_p$, $A = g^a$
  - B chooses $b \leftarrow \mathbb{Z}_p$, $B = g^b$

Depending on i, in the second message, B either sends

- $\widetilde{B} = B$ if $c = 0$ or
- $\widetilde{B} = AB$ if $c = 1$

Now A calculates both

- $k_0 = (\widetilde{B})^a$
- $k_1 = (\frac{\widetilde{B}}{A})^a$

and sends both $e_0 \leftarrow E_{k_0}(m_0)$ and $e_1 \leftarrow E_{k_1}(m_1)$

Info Threshold Decryption and Threshold Signatures

- A threshold public key encryption system is a
  - public key system where the private key is
    - distributed among n decryption servers so that
    - at least k servers are needed for decryption
- In a threshold encryption system an entity
  - called the combiner
    - has a ciphertext c that it wishes to decrypt
- The combiner sends c to the decryption servers
  - and receives partial decryption shares
    - from at least k out of the n decryption servers
- It then combines these k partial decryptions
  - into a complete decryption of c
- Ideally, there is no other interaction in the system
  - namely the servers need not talk to each other during decryption
- Such threshold systems are called non-interactive

Dining cryptographers

- David Chaum proposed 1988 the Dining cryptographers
  - showing it is possible to send anonymous messages with
    - unconditional sender and recipient untraceability
- Cryptographers A_i for $i = 1, 2, \ldots, n$ around a table for dinner
  - A_i has a secret s_i
- Collectively they want to calculate
  - $s_1 \oplus s_2 \oplus \ldots \oplus s_i \ldots \oplus s_n$
  - $\Sigma_i \; s_i \pmod 2$

Dining cryptographers, "Toy Use Case":

- The waiter informs them that the meal
  - has been paid for by someone
    - who could be one of the cryptographers or their boss
- The cryptographers respect each other's right to
  - make an anonymous payment
    - but want to find out whether the boss paid
    - (The boss has no privacy right here)

Dining cryptographers

- Notice that each cryptographer has a secret s_i
  - which is 1 if he paid for the meal and 0 else
- So they decide to execute a two-stage protocol
- In the first phase
  - Each two cryptographers A_i, A_{i+1} sitting next to each other
    - establish a shared random one-bit secret $b_{i,i+1}$
    - so that only those two cryptographers know the outcome
  - Example with 3 cryptographers:
    - A_1, A_2 share secret $b_{1,2} = 1$
    - A_2, A_3 share $b_{2,3} = 0$
    - A_3, A_1 share $b_{3,4} = 1$

Dining cryptographers: one solution

- Now each cryptographer
  - publicly announces the bit
    - $a_i = s_i \oplus b_{i,i-1} \oplus b_{i,i+1}$
    - where $i-1, i+1$ are mod n

Then $\Sigma_i s_i = \Sigma_i a_i$, because in the second sum

- each of the numbers $b_{i,i-1}$ appears twice (in a_i and in a_{i+1})
  - and therefore cancel out

Thus the sum $\Sigma_i a_i$ reveals if one of the s_i is one

- That is, one of the cryptographers paid

Dining cryptographers: one solution

- In other words
  - if A_i didn't pay for the meal
    - he shows the xor of
    - the two shared bits he holds with his neighbours
  - if he did pay for the meal
    - the opposite of that xor
- In the example (above)
  - if none of the cryptographers paid, then
    - A_1 would announce $b_1 = 0 \oplus 1 \oplus 0 = 1$,
    - A_2 would announce $b_2 = 0 \oplus 1 \oplus 1 = 0$, and
    - A_3 would announce $b_3 = 0 \oplus 0 \oplus 1 = 1$
- On the other hand
  - if A_1 paid, he would announce
    - $b_2 = 1 \oplus 1 \oplus 1 = 1$

Dining cryptographers: one solution

- Notice that the xor of
  - all the announced bits b_i is 0
    - iff none of the cryptographers paid
    - so the boss must have paid
- Otherwise if the xor of
  - all the announced bits b_i is 1
    - then one of the cryptographers paid
    - but his identity remains unknown
    - to anybody, including the other cryptographers
- Anonymous communication networks
  - based on this problem are often known as DC-nets

Secure Multi-Party Computations

Assume n parties $P_1, \ldots, P_n$

- each one in possession of a (secret) value x_i

The parties want to calculate a set of functions

- $\bar{f}(\bar{x}) = f_1(x_1, \ldots, x_n, r), \ldots, f_n(x_1, \ldots, x_n, r)$ with $r \leftarrow \mathscr{D}$
  - over their data, $(x_1, \ldots, x_n)$,
    - plus perhaps a (common) random input r, with:

Requirements

- Party P_i should learn the result $f_i(x_1, \ldots, x_n, r)$ and
  - should learn nothing else
- No external to the protocol (eavesdropper)
  - should learn anything
- This should hold even
  - if an arbitrary subset of the parties maliciously deviates from the protocol

Secure Multi-Party Computations

This can be easily done if the parties have

- direct, unrestricted and secure access to an
  - "angelic" trusted third party (T3P)

Then: each party P_i sends the input x_i to the T3P

- over an ideal secure channel
  - no one can read or modify this value

The T3P computes $y_1 = f_1(x_1, \ldots, x_n, r), \ldots, y_n = f_n(x_1, \ldots, x_n, r)$

- sends y_i to P_i
  - (over the secure channel)

Secure Multi-Party Computations

Secure multi-party computations do the same thing

- but not relying on a third party
  - but rather only on cryptographical methods

Def: a protocol π securely realizes $\bar{f}(\bar{x})$ if

running π emulates

- an ideal process where
  - all parties secretly provide inputs to an trusted party
  - which computes $\bar{f}$ and returns the outputs to the parties
- and any "harm" done by a ppt adversary
  - in the real execution ofπ
- could have been done by
  - a ppt in the ideal process

Secure Multi-Party Computations

- The T3P solution provides not only security against individual cheaters
  - also ensures security if several parties are colluding throughout the entire execution
  - If some set B of parties collude
  - then the parties in that set learn the union of what they each learn individually
  - but nothing more
- The solution using a trusted party is "the best one could hope for"
  - (if the T3P is really trustworthy)
  - and we will therefore take this as our "ideal world"
- In the real world, in contrast
  - there may not exist any trusted parties that all the players agree upon

Secure Multi-Party Computations

- Protocols for secure computation should provide a way for
- $P_1, \ldots, P_n$ to achieve the security guarantees of the ideal world without the T3P
- Roughly speaking
  - a protocol is "secure" if the actions of any colluding parties in the real world can be emulated by those same parties in the ideal world
- But let us generalize the model a bit by introducing randomicity and introducing a "*don't care*" notion

Secure Multi-Party Computations

- The concept of Secure Multi-Party Computation generalizes
  - confidentiality and integrity of data

Example/Exercise

Describe the problem of

- secure (confidentiality and integrity protected) communication
  - from P_1 to P_2

as a MPC problem of computing the function $f_?(x_?) = x_?$

Secure Multi-Party Computations

Recall we have n parties

- each one in possession of a (secret) value x_i

They want to calculate

- $f_1(x_1, \ldots, x_n, r), \ldots, f_n(x_1, \ldots, x_n, r)$ with $r \leftarrow \mathscr{D}$
  - (with some additional random input r)
- and party i gets exactly the result of f_i

Shorthand $F(x_1, x_2, \ldots, x_n) :=$

$(f_1(x_1, x_2, \ldots, x_n, r), \ldots, f_n(x_1, x_2, \ldots, x_n, r))$

- F is the function that
  - takes all the inputs from all parties and
  - calculates all the outputs for them
- Info (Remark: F is non-deterministic, while each f_i is)

Secure Multi-Party Computations

- The concept of Secure Multi-Party Computation is
  - very general and
  - very strong

Examples

- $F(x_1, \ldots, x_n) = x_1 + \ldots + x_n$ a simple sum function
  - where all parties get the same value
- $F(x_1, \ldots, x_n) = MAX(x_1, \ldots, x_n)$ a max-value function
- $F(-, \ldots, -) = r \leftarrow \mathcal{D}$ a simple coin toss function
  - Here the main requirement is that the output remains unbiased in spite of any malicious behavior

Secure Multi-Party Computations

Examples

- $F((x_0, x_1), b) = (-, x_b(b \leftarrow \{0, 1\}))$
  - This is 1-of-2 oblivious transfer:
  - Party 2 learns one of two values that party 1 had
  - and party 1 doesn't know which value party 2 learned
- $F_R((x, w), -) = (-, (x, R(x, w)))$ where $R(x, w)$ is a binary relation
  - This is a modelling of Zero-Knowledge (we have correctness and soundness)
  - This is also a proof of knowledge of a witness

MPC Yao's Garbled Circuits

Assume parties $P1$ and P_2 share a Boolean Circuit f_c for $f = (f_1, f_2)$

- With inputs i_1 that will be provided by $P1$ and
- i_2 to be provided by $P2$

This circuit is in "cleartext":

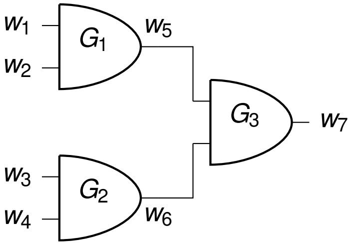
- It has a finite number of wires $w_1, w_2, \ldots$ and gates $g_1, g_2 \ldots$
  - Each gate g_i is given by a "truth table" on three wires
- In a moment we will explain
  - how wires and gates can be "garbled",
    - constructing a "garbled circuit"
  - how to evaluate a "garbled circuit":
    - given garbled wire values
    - how to calculate the garbled output wire

The high-level view of Yao's construction is given in the next slide

MPC Yao's Garbled Circuits, Overview

1. $P1$ garbles each wire and each gate of the clear-text circuit f_c
   - except the output bits of f_2
   - creating garbled circuit f_g
2. $P1$ sends f_g and the garbled values for his inputs i_1
3. $P2$ uses OT to get the garbled values of his inputs i_2 in f_g
4. $P2$ calculates f_g with the garbled versions of i_1 and i_2
   - obtains his output f_1 and sends to $P1$ the garbled values for f_2

MPC A Circuit in Cleartext



Where the gates are given by tables, i.e for G_3 (an or-gate):

w_5	w_6	w_7
0	0	0
0	1	1
1	0	1
1	1	1

Figure: Gate G_3: $w_7 = w_5 \vee w_6$ in cleartext

Garbling the wires

For each wire number $w := 1, 2, 3, \ldots$ and possible value on the wire ($v := 0, 1$)

- choose a random number called the "random encoding" e_w^v
  - Thus e_5^1 encodes the fact that "wire 5 has value 1"

The first party has thus a (secret) "translation table" for all possible values for all wires:

wire	*value*	*encoded − value*
w_1	0	e_1^0
w_1	1	e_1^1
w_2	0	e_2^0
w_2	1	e_2^1
...	...	...

Figure: Table: Encrypted Wire Values for all wires and all possible values

Garbling a Circuit

w_5	w_6	w_7
e_5^0	e_6^0	e_7^0
e_5^0	e_6^1	e_7^1
e_5^1	e_6^0	e_7^1
e_5^1	e_6^1	e_7^1

(a) Gate G_3 using encrypted wires $w_7 = w_5 \vee w_6$

garbled gate
$h(e_5^1 \| \| e_6^0 \| \| g_3) \oplus e_7^1$
$h(e_5^0 \| \| e_6^0 \| \| g_3) \oplus e_7^0$
$h(e_5^0 \| \| e_6^1 \| \| g_3) \oplus e_7^1$
$h(e_5^1 \| \| e_6^1 \| \| g_3) \oplus e_7^1$

(b) Garbled G_3 $(w_7 = w_5 \vee w_6)$

Figure: Garbled Computation table for G_3

Calculating a Garbled Circuit

What is a garbled circuit?

It is a circuit, like one in cleartext:

- contains wires and gates
- contains tables representing the gates
  - but those tables are not "truth tables" (cleartext)
  - they are "Garbled Computation Tables" as in the previous slide
    - (rows have the form, say: $h(e_5^0||e_6^0||g_3) \oplus e_7^0$)

How can you calculate a garbled circuit?

As for a normal circuit, you calculate each gate at a time in sequence

- but you work with garbled wire values, not with Booleans

Given, as input, the garbled wire values of a garbled gate

- In our example, given for instance e_5^0, e_6^1
- It is easy to calculate $h(e_5^0||e_6^1||g_3)$

Calculating a Garbled Circuit

There is only one problem:

- the party who is calculating the garbled gate g_3
  - has some "strange" values as inputs, say e_5^0 and e_6^1
    - but they are just some random looking numbers
  - he does not know that they are e_5^0 and e_6^1
    - they could be e_5^1 and e_6^0 (or any of the 4 combinations)
- thus, he does not know if he "is" in the row $h(e_5^0||e_6^1||g_3) \oplus e_7^1$
  - or in another row, say $h(e_5^1||e_6^1||g_3) \oplus e_7^1$
    - if he uses this row, he gets an incorrect answer for the garbled value of w_7
- You need some redundancy / markers / signals (see class)

Concepts: Non-transferable Proof; Interactive protocol

An interactive protocol

Between 2 or more parties,

- requires both to be on-line simultaneously

Encrypting or signing a message is (typically) non-interactive:

- Say in secure email the sender is not necessarily on-line
  - when the email message is decrypted or the signature is verified

Challange.response or ZKP (like Schnorr, see below)

- are typically interactive

Concepts: Non-transferable Proof; Interactive protocol

A proof is "non-transferrable"

If it convinces "me" (the verifier)

- but the proof, no matter how I record it
  - will not convince other people
    - because it is easy for me to fake such proofs

It may be generated in

- an interactive or
- non-interactive protocol

Concepts: Non-transferable Proof; Interactive protocol

- Example: a MAC (message authn code) is produced in
  - a non-interactive protocol
- the proof is non-transferrable
  - although it convinces me that it was generated
    - by the only other entity that knows the key

Example

Key agreement is (typically) interactive

- but it is not a proof
  - you may have key agreements based on shared keys
  - or non-authenticated key agreements

Interactive Zero-Knowledge Proofs

- An interactive proof
  - transfers the conviction to the verifier
    - that the claimed statement is true
  - but does not leak any further information
    - in particular, it does not create a transferable proof
    - that could convince anybody else
- The interactive proof is zero-knowledge
  - if the "transcript" of the proof
    - could have been constructed by anybody
- We say: anybody could simulate any protocol transcript
  - without interacting with the prover
- Of course, the transcript of the conversation
  - is not convincing for any other party

Proof of knowledge / proof of possession

- Given a "public number" z
  - s.t there is a "secret number" x
    - in a particular relation to z: $R(x, z)$
- Suppose P can be identified as the only entity
  - who knows the secret x
- We want a method that allows P to
  - prove that he knows the secret x
    - without disclosing *anything* about x
    - except that $R(x, z)$
- Example: Given
  - a group $G = \langle g \rangle$, of order p,
  - a generator g and
  - $z \in G$ (say, the token of P)
- P claims that he knows the secret x with $z = g^x$
- A simple proof of this knowledge is Schnorr's identification scheme

Schnorr's identification scheme

- P claims to know x s.t $g^x = z$
- P chooses randomly k

$$P \xrightarrow{t=g^k} V$$

$$P \xleftarrow{c} V$$

$$P \xrightarrow{r=cx+k} V$$

- V accepts if $g^r = tz^c$
- If P knows the secret x
  - Then: $g^x = z$, $r = cx + k$, and $t = g^k$
  - $\Rightarrow g^r = g^k g^{cx} = t(g^x)^c = tz^c$
    - and therefore V accepts

Properties of Schnorr's identification scheme

- Schnorr's identification scheme has 3 key properties:
- The proof presented to V
  - cannot be used offline to demonstrate to anybody
    - that P (or anyone) knows the secret x:
  - in fact anybody could present a transcript
    - simulating having carried out a successful exchange
- It is secure
  - If V runs the protocol correctly
    - and P does not know the secret x
  - Then the probability that
    - P is able to answer the challenge c (message 2) correctly
    - is negligible
- The protocol discloses
  - absolutely no information about the secret
    - to V
    - nor to anybody else

Info Subgroups

- A subset G of the group $\widehat{G}$ is a *subgroup* of $\widehat{G}$
  - written as $G \leq \widehat{G}$ iff G is itself a group with respect to the operation of $\widehat{G}$

Info Lagrange's Theorem: H subgroup of $G \Rightarrow |H| \mid |G|$

- Proof: Let G be a group
  - H be a subgroup of G. For each $x \in G$ consider

$$xH := \{x \circ h \mid h \in H\}$$

- We claim that the sets xH are all of the size of H and form a partition of G
  - It follows immediately that the size of H divides the size of G

Info Lagrange's Theorem: H subgroup of $G \Rightarrow |H| \mid |G|$

- Two observations:
- For $x, y \in G$ either xH and yH are equal or disjoint:
  - If $xH \cap yH \neq \emptyset$ then there are $h_1, h_2 \in H$ such that
    - $x \circ h_1 = y \circ h_2$ and thus for any $h \in H$ it follows
    - $x \circ h = y \circ h_2 \circ h_1^{-1} \circ h \in yH$
  - Thus $xH \subseteq yH$ and by symmetry $xH = yH$
- The function

$$(x \circ \cdot) : H \rightarrow xH,$$
$$h \mapsto x \circ h$$

- is 1-1 ($x \circ h_1 = x \circ h_2 \Rightarrow h_1 = h_2$
- cancelling x) and onto (by definition of xH)

Info Exercise on Lagrange's Theorem

- Let G be a group
  - H be a subgroup of G
  - $x \in G$ and $xH := \{x \cdot h \mid h \in H\}$ as before
- For every $x, y \in G$ let
  - $x \sim y :\Leftrightarrow xH = yH$
  - $x \sim y \Leftrightarrow x^{-1}y \in H$
- $\sim$ is an equivalence relation and the equivalence classes are precisely the sets xH
  - Exercise: In the particular case of $G = (\mathbb{Z}, +)$ and $H = n\mathbb{Z}$ the subgroup of multiples of n
  - calculate $\sim$ and $G/\sim$

Info Cyclic Groups

- Starting from any element g in any group $\widehat{G}$
  - consider the set of all powers of $g \in \widehat{G}$
- This is a subgroup of $\widehat{G}$:
- it is denoted $\langle g \rangle$ and called the *subgroup generated by g*
- Note that this group $\langle g \rangle$ is always commutative
  - even if $\widehat{G}$ is not

Info Subgroups, Cyclic Groups, Order of elements

- If $\langle g \rangle$ is finite
  - its size is called *the order of g* (and *the order of the subgroup* $\langle g \rangle$)
- Thus $\text{ord}(g) = \text{ord}(\langle g \rangle) = |\langle g \rangle| = \min\{i \mid g^i = e\}$
- A group G is cyclic if it has an element g s.th
- $G = \langle g \rangle$
- Any finite cyclic group of order n is therefore of the form:
- $G = \{e, \underbrace{g}, \underbrace{g \circ g}, \underbrace{g \circ g \circ g}, \ldots, \underbrace{g \circ g \circ g \circ g \circ \ldots \circ g \ (n-1 \text{ times})}\}$
- $= \{e, \quad g, \quad g^2 \quad , \quad g^3 \quad , \ldots, \quad g^{n-1} \quad \}$
- Notice that any two cyclic groups of the same order are isomorphic
- In particular any cyclic groups is isomorphic to some "simple group" of the form $(\mathbb{Z}_n, +_n)$ (next slide)

Info A "simple" group

- $\mathbb{Z}_n = \{0, 1, 2, 3, \ldots n-1\}$ with $+_n$ the sum modulo n as operation is a group for each $n \in \mathbb{N}$
- The size of the group is n
- This is an example of a "simple group" – that is a group where all interesting operations are easy to evaluate – but
  - as we will see
  - it is isomorphic to some *complex groups* where corresponding operations may be quite difficult
- It may sound strange that operations in one group G_1 are simple and the "same" operations in an isomorphic group G_2 are difficult
- but it is possible that in one direction the isomorphism
- $G_1 \rightarrow G_2$ is easy to calculate (say, using exponentiation)
  - while the reverse isomorphism $G_2 \rightarrow G_1$ may be difficult or even infeasible to calculate (requiring the computation of a discrete logarithm)

Info Examples of Groups

- The following are groups:
- $\mathbb{Z}_p^*$: for some prime p
  - is the set of elements
    - $\{1, 2, 3, \ldots p-1\}$ under the operation multiplication The size of the group is $p-1$
- $\mathbb{Z}_7$: consists of $\{1, 2, 3, 4, 5, 6\}$. For instance
  - $5 * 5 \equiv_7 25 equiv_7 4$
    - The inverse can be derived similarly
  - for instance 3^{-1} is represented by 5 since $3 * 5 \equiv_7 15 \equiv_7 1$
- $G = \{1, 2, 4\}$ is a group under the operation multiplication modulo 7
- $G = \{1, 2, 4, 6\}$ is not a group under the operation multiplication modulo 7 because it does not obey the closure property:
  - $2 * 6 \pmod 7 \notin G$
- Elliptic Curve groups

$\mathbb{Z}_n^*$

- For $a \in \mathbb{Z}_n^*$, let the order of a be:

$$\mathrm{ord}_{\mathbb{Z}_n^*}(a) := \min\{k \mid a^k \equiv_n 1\}$$

Info Fermat's Theorem, Euler's Theorem

- Defs (recall): Order, generator
  - If G is finite, then
    - $\langle g \rangle := \{g^i : i \in \mathbb{Z}\}$
    - is also finite; the size is
- $|\langle g \rangle| = \text{order}(g) := \min_i \{g^i = 1\}$
- Thus $\langle g \rangle = \{1, g, g^2, g^3, \ldots, g^{\text{order}(g)-1}\}$
- An element $g \in G$ is called a *generator* of G if
- $\langle g \rangle = G$ or equivalently, the order of g is $|G|$

Info Fermat's Theorem, Euler's Theorem

- Euler's Theorem
  - The order of every element $g \in G$ divides $|G|$
  - This follows from Lagrange's Theorem, since the size of the subgroup
  - $\langle g \rangle$ must divide the size of the group
- A simple consequence is:
  - Fermat's Theorem For every prime p and $g \in \mathbb{N}$,
    - $g^{p-1} = 1 \pmod{p}$

Info Application: generating random primes

- Suppose we want to generate a large random prime p of length 1024 bits (i.e. $p \approx 2^{1024}$)
- Choose a random integer $p \in [2^{1024}, 2^{1025} - 1]$
- Test if $2^{p-1} = 1$ in $\mathbb{Z}_p$
  - If yes, done
  - If not, try another p
- This is a simple algorithm, but not the best

$$\Pr[p \text{ passes the test but is not prime}] < 2^{-60}$$